Lab 5 – Contrôle d'Exécution

Les boucles

Souvent, vous voudrez faire plusieurs choses en une seule tâche, comme

- Créer beaucoup d'utilisateurs
- Répéter une requête jusqu'à ce qu'un certain résultat soit atteint.

Ansible prend en charge les boucles pour itérer sur un ensemble de valeurs et évite aux administrateurs d'écrire des tâches répétitives utilisant les mêmes modules.

Ansible prend en charge trois principaux types de boucles: les boucles simples, la liste de hachages et les boucles imbriquées. Dans cet atelier, nous examinerons rapidement les deux premiers types.

Ansible 2.5 a apporté une nouvelle manière plus simple de travailler avec des boucles: le mot-clé loop. Avant 2.5, Ansible utilisait principalement les mots-clés *with_<lookup>* pour créer des boucles, le nouveau mot-clé loop est fondamentalement analogue à with\list. Vous trouverez de nombreux exemples Ansible utilisant with\items ou with\nested sur Internet.

Boucles simples

Les boucles simples sont une liste d'éléments sur lesquels Ansible itère. Ils sont définis en fournissant une liste d'éléments au mot-clé **loop**. Créez le Playbook suivant `loop1.yml` et exécutez-le:

```
---
- name: Loop demo
  hosts: centos01
  tasks:
  - name: Check if service is started
    service:
      name: "{{ item }}"
      state: started
    loop:
    - httpd
    - sshd
```

La liste des éléments à parcourir peut également être fournie sous forme de liste dans la section **vars** ou dans un fichier. Dans cet exemple, le tableau est appelé check_services. Créez ce Playbook en tant que `loop2.yml` et exécutez-le, la sortie doit être la même:

```
---
- name: Loop demo
  hosts: centos01
  vars:
    check_services:
```

```
    - httpd
    - sshd
  tasks:
  - name: Check if service is started
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ check_services }}"
```

Les Hashs

Vous pouvez également parcourir une liste de **hashs** pour des données plus complexes. Le Playbook suivant montre comment une liste de **hashs** avec des paires clé-valeur est transmise au module **user**. Créez `loop3.yml` et exécutez-le:

```
---
- name: Hash demo
  hosts: centos01
  become: yes
  tasks:
  - name: Create users from hash
    user:
      name: "{{ item.name }}"
      state: present
      groups: "{{ item.groups }}"
    loop:
    - { name: 'jane', groups: 'wheel' }
    - { name: 'joe', groups: 'root' }
```

Conditionnels Ansible

Ansible peut utiliser des conditions pour exécuter des tâches lorsque certaines conditions sont remplies.

Pour implémenter une condition, l'instruction **when** doit être utilisée, suivie de la condition à tester. La condition est exprimée en utilisant l'un des opérateurs disponibles comme par exemple pour comparaison:

== Compare deux objets pour l'égalité. != Compare deux objets pour l'inégalité. > True si le côté gauche est plus grand que le côté droit. >= True si le côté gauche est supérieur ou égal au côté droit. < True si le côté gauche est plus bas que le côté droit. <= True si le côté gauche est inférieur ou égal au côté droit.

Utiliser des conditions

Par exemple, vous souhaitez installer un serveur FTP, mais uniquement sur des hôtes qui ne sont pas en production car le protocole n'est pas sécurisé.

En tant qu'utilisateur, créez le Playbook `ftpserver.yml`, exécutez-le et examinez la sortie :

```
---
- name: Install insecure FTP server as long as not in production
  hosts: all
  become: yes
  tasks:
  - name: Install FTP server if not in production
        yum:
          name: vsftpd
          state: latest
          when: stage != "prod"
```

Résultat attendu : la tâche est ignorée sur *centos01* car la variable *stage* est définie sur prod:

[...] TASK [Install FTP server if not in production] ********** skipping: [centos01] changed: [centos03] changed: [centos02] changed: [ubuntu01] [...]

Challenge Lab: Fact en conditionnel

Essayons autre chose. Vous avez probablement remarqué *centos01* et *ubuntu01* ont différentes quantités de RAM. Sinon, regardez à nouveau les facts:

```
$ ansible all -m setup -a 'filter=ansible_*_mb'
```

Écrivez un Playbook `mariadb.yml` qui installe *MariaDB* mais seulement si l'hôte a plus de, disons, ***900Mo*** de RAM.

- Trouvez le fact pour `memtotal` en Mo (regardez la sortie de la commande ad-hoc et n'hésitez pas à utiliser "grep").
- Utilisez ce Playbook comme modèle et créez l'instruction **when** en remplaçant les espaces réservés en majuscules :

```
---
- name: MariaDB server installation
  hosts: all
  become: yes
  tasks:
  - name: Install latest MariaDB server when host RAM greater 9000 MB
    yum:
      name: mariadb-server
      state: latest
      when: FACT COMPARISON\OPERATOR NUMBER
```

Exécutez le Playbook. En conséquence, la tâche d'installation doit être ignorée sur host2.

Solution

```
---
- name: MariaDB server installation
  hosts: all
  become: yes
  tasks:
  - name: Install latest MariaDB server when host RAM greater 9000 MB
    yum:
      name: mariadb-server
      state: latest
      when: ansible\memtotal\mb > 900
```

Ansible Handlers

Parfois, lorsqu'une tâche modifie le système, une autre tâche devra être exécutée. Par exemple, une modification du fichier de configuration d'un service peut alors exiger que le service soit rechargé afin que la configuration modifiée prenne effet.

Ici les **handlers** d'Ansible entrent en jeu. Les **handlers** peuvent être considérés comme des tâches inactives qui ne sont déclenchées que lorsqu'elles sont explicitement appelées à l'aide de l'instruction "***notify***".

À titre d'exemple, écrivons un Playbook qui :

- gère le fichier de configuration d'Apache ***httpd.conf*** sur toutes les hôtes du groupe *webserver*
- redémarre Apache lorsque le fichier est changé

Nous avons d'abord besoin du fichier que Ansible va déployer, prenons simplement celui de master :

```
[master]$ cp /etc/httpd/conf/httpd.conf .
```

Créons maintenant le Playbook httpd\conf.yml:

```
---
- name: manage httpd.conf
  hosts: centos
  become: yes
  tasks:
  - name: Copy Apache configuration file
    copy:
      src: httpd.conf
      dest: /etc/httpd/conf/
      notify:
        - restart_apache
  handlers:
  - name: restart_apache
```

```
    service:
      name: httpd
      state: restarted
```

Alors, quoi de neuf ici ?

- La section ***notify*** appelle le **handler** uniquement lorsque la tâche de copie a modifié le fichier.
- La section ***handlers*** définit une tâche qui n'est exécutée que sur notification.

Exécutez le Playbook. Nous n'avons encore rien changé dans le fichier donc il ne devrait y avoir aucune ligne *changed* dans la sortie et bien sûr le handler ne se déclenche pas.

- Maintenant, changez la ligne ***Listen 80*** dans httpd.conf en :

```
Listen 8080
```

- Exécutez à nouveau le Playbook. Maintenant, la sortie d'Ansible devrait être beaucoup plus intéressante :
  - httpd.conf doit être copié
  - Le ***handler*** doit avoir redémarré Apache

Apache devrait maintenant écouter sur le port 8080. Assez facile à vérifier :

```
[master]$ curl node1.local
curl: (7) Failed connect to node1.local:80; Connection refused
```

```
[master]$ curl node1.local:8080
<body>
<h1>This is a production webserver, take care!</h1>
</body>
```